optimizing a certified proof checker for a large-scale computer-generated proof

luís cruz-filipe

(joint work with peter schneider-kamp)

department of mathematics and computer science
university of southern denmark

cicm 2015
july 16th, 2015

outline

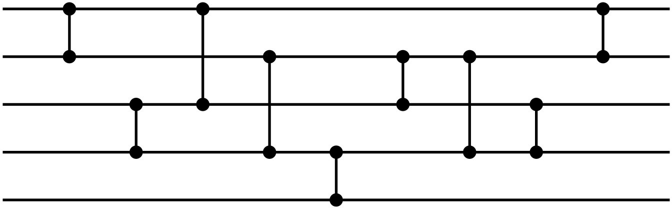
sorting networks in a nutshell

checking the result

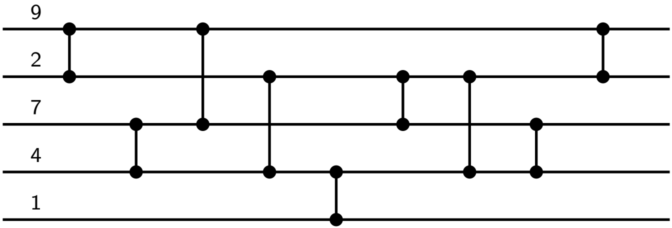
making the checker work

conclusions & future work

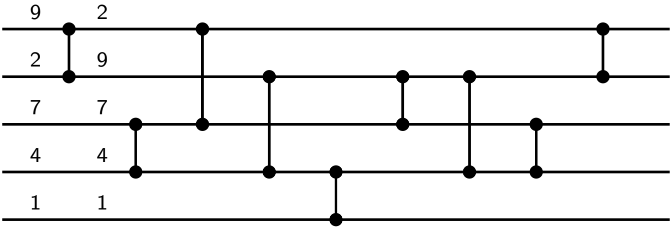
a sorting network



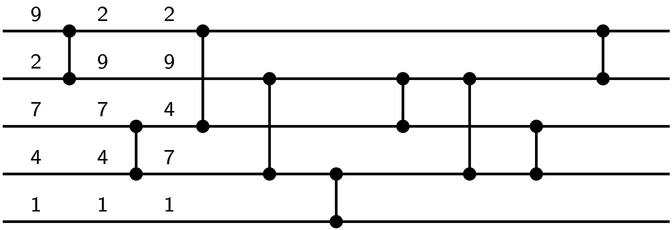
a sorting network



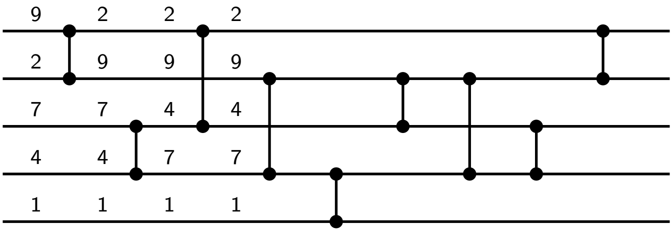
a sorting network



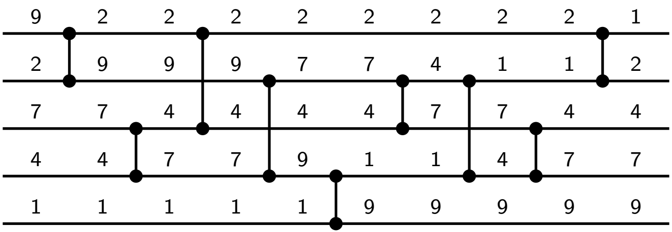
a sorting network



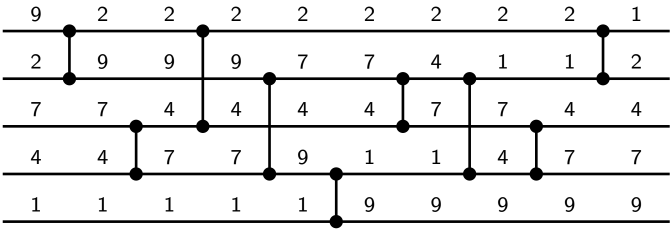
a sorting network



a sorting network

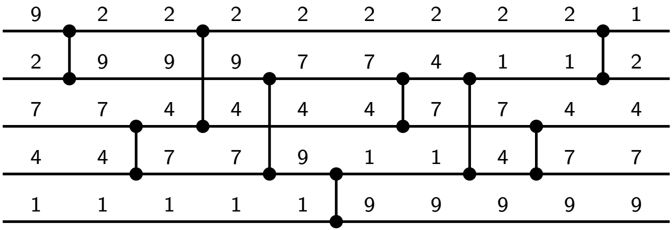


a sorting network



size this net has 5 *channels* and 9 *comparators*

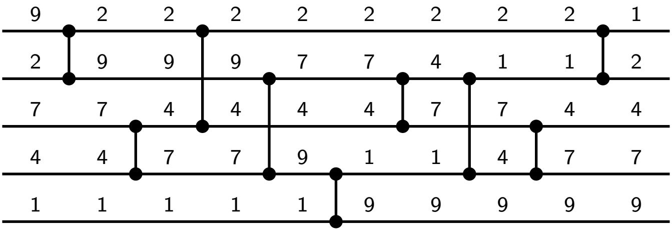
a sorting network



size this net has 5 *channels* and 9 *comparators*

more info see d.e. knuth, *the art of computer programming*, vol. 3

a sorting network



size — this net has 5 *channels* and 9 *comparators*

more info — see d.e. knuth, *the art of computer programming*, vol. 3

the optimal size problem — what is the minimal number of *comparators* in a sorting network on n channels (s_n)?

history

optimal size

s_n: minimal number of *comparisons* to sort n inputs

knuth 1973

n	1	2	3	4	5	6	7	8	9	10
s_n	0	1	3	5	9	12	16	19	25 23	29 27

n	11	12	13	14	15	16	17
s_n	35 31	39 35	45 39	51 43	56 47	60 51	73 56

- values for $n \leq 4$ from information theory
- values for $n = 5$ and $n = 7$ by exhaustive case analysis

knuth $s_n \geq s_{n-1} + 3$ $\rightsquigarrow$ values for $n = 6, 8$

van voorhis $s_n \geq s_{n-1} + \lg(n)$ $\rightsquigarrow$ other lower bounds

history

optimal size

yours truly 2014

s_n: minimal number of *comparisons* to sort n inputs

n	1	2	3	4	5	6	7	8	9	10
s_n	0	1	3	5	9	12	16	19	**25**	**29**

n	11	12	13	14	15	16	17
s_n	35 **33**	39 **37**	45 **41**	51 **45**	56 **49**	60 **53**	73 **58**

- generate-and-prune algorithm
- intensive parallel computing
- $\sim$ 16 years of cpu time to compute s_9

history

optimal size

s_n: minimal number of *comparisons* to sort n inputs

yours truly 2014

n	1	2	3	4	5	6	7	8	9	10
s_n	0	1	3	5	9	12	16	19	**25**	**29**

n	11	12	13	14	15	16	17
s_n	35 **33**	39 **37**	45 **41**	51 **45**	56 **49**	60 **53**	73 **58**

- generate-and-prune algorithm
- intensive parallel computing
- $\sim$ 16 years of cpu time to compute s_9

- but how do we know that these results are correct?

outline

sorting networks

comparator	sequence of *comparators* (i, j) with $1 \leq i < j \leq n$
network	n is the number of channels

sorting networks

comparator network — sequence of *comparators* (i, j) with $1 \leq i < j \leq n$
n is the number of channels

0/1 lemma (knuth 1973) — C is a sorting network on n channels iff C sorts all inputs in $\{0,1\}^n$

sorting networks

comparator network — sequence of *comparators* (i, j) with $1 \leq i < j \leq n$
n is the number of channels

0/1 lemma (knuth 1973) — C is a sorting network on n channels iff C sorts all inputs in $\{0, 1\}^n$

output — $C(\vec{x})$ denotes the *output* of C on $\vec{x} = x_1 \dots x_n$

sorting network — $C(\vec{x})$ is sorted for every input $\vec{x}$

sorting networks

comparator network	sequence of *comparators* (i, j) with $1 \leq i < j \leq n$ n is the number of channels
0/1 lemma (knuth 1973)	C is a sorting network on n channels iff C sorts all inputs in $\{0,1\}^n$
output	$C(\vec{x})$ denotes the *output* of C on $\vec{x} = x_1 \dots x_n$
sorting network	$C(\vec{x})$ is sorted for every input $\vec{x}$
typical result	"C is a sorting network" is decidable

- program extraction $\rightsquigarrow$ haskell program (tests all inputs)
- nearly best possible algorithm (known result)
- short formalization ($\sim$ 35 lemmas)

the key result

output lemma (parberry 1991)

if outputs(C) $\subseteq$ outputs(C') and $C'; N$ is a sorting network, then $C; N$ is a sorting network

the key result

output lemma (parberry 1991)

if outputs(C) $\subseteq$ outputs(C') and $C'; N$ is a sorting network, then $C; N$ is a sorting network

permuted output lemma

if π(outputs(C)) $\subseteq$ outputs(C') for some permutation π and C' extends to a sorting network, then C extends to a sorting network

the key result

output lemma (parberry 1991)

if outputs(C) $\subseteq$ outputs(C') and $C'; N$ is a sorting network, then $C; N$ is a sorting network

permuted output lemma

if π(outputs(C)) $\subseteq$ outputs(C') for some permutation π and C' extends to a sorting network, then C extends to a sorting network

definition

$C \preceq_\pi C'$ if π(outputs(C)) $\subseteq$ outputs(C')
$C \preceq C'$ if $C \preceq_\pi C'$ for some permutation π

the algorithm

init set $R_0^n = \{\emptyset\}$ and $k = 0$

repeat until $k > 1$ and $|R_k^n| = 1$

generate N_{k+1}^n extend each net in R_k^n by one comparator in all possible ways

prune to R_{k+1}^n keep only one element from each minimal equivalence class w.r.t. $\preceq^T$

step increase k

the algorithm

init set $R_0^n = \{\emptyset\}$ and $k = 0$

repeat until $k > 1$ and $|R_k^n| = 1$

generate N_{k+1}^n extend each net in R_k^n by one comparator in all possible ways

prune to R_{k+1}^n keep only one element from each minimal equivalence class w.r.t. $\preceq^T$

step increase k

pruning

- quadratic step
- inner loop searches among all permutations typically fails
- record successful subsumptions

the algorithm

init set $R_0^n = \{\emptyset\}$ and $k = 0$

repeat until $k > 1$ and $|R_k^n| = 1$

generate N_{k+1}^n extend each net in R_k^n by one comparator in all possible ways

prune to R_{k+1}^n keep only one element from each minimal equivalence class w.r.t. $\preceq^T$

step increase k

certified checker using recorded subsumptions as an oracle

- replace pruning cycle by oracle calls
- skeptic approach towards oracle
- use program extraction
- verifies all cases up to s_8, requires $\sim$18 years for s_9...

checker soundness

```
Definition Oracle := list (comp_net * comp_net * (list nat)).

Inductive Answer : Set :=
  | yes : nat -> nat -> Answer
  | no : forall m n:nat, forall R:list comp_net,
         NoDup R ->
        (forall C, In C R -> length C = n) ->
        (forall C, In C R -> standard m C) -> Answer
  | maybe : Answer.

Fixpoint Generate_and_Prune (m n:nat) (O:list Oracle) :
   Answer.

Theorem GP_no : forall m n O R HR0 HR1 HR2,
   Generate_and_Prune m n O = no m n R HR0 HR1 HR2 ->
   forall C, sort_net m C -> length C > n.

Theorem GP_yes : forall m n O k,
   Generate_and_Prune m n O = yes m k ->
  (forall C, sort_net m C -> length C >= k) /\
   exists C, sort_net m C /\ length C = k.
```

outline

sorting networks in a nutshell

checking the result

making the checker work

conclusions & future work

an offline oracle

typical approach

- call oracle to solve difficult tasks
- check result
- oracle is online, waiting for the next problem

an offline oracle

typical approach

- call oracle to solve difficult tasks
- check result
- oracle is online, waiting for the next problem

in our case

- oracle is pre-computed (offline)
- information from oracle guides algorithm
- potential for optimizations

improving the pruning step

old algorithm while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. check that C, C' are in the current set
3. remove C' from the current set

(laziness performs the last two steps together)

improving the pruning step

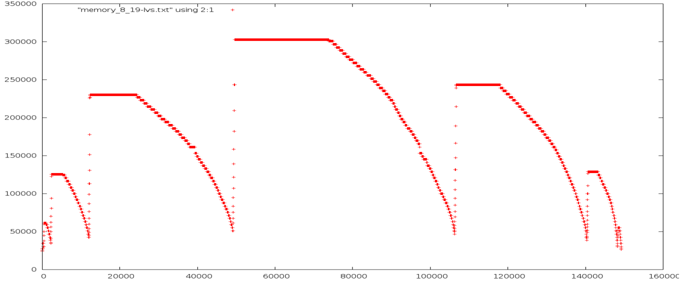
old algorithm while oracle has a next subsumption $C \preceq_\pi C'$

1 check that $C \preceq_\pi C'$

2 check that C, C' are in the current set

3 remove C' from the current set

(laziness performs the last two steps together)



improving the pruning step

old algorithm

while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. check that C, C' are in the current set
3. remove C' from the current set

(laziness performs the last two steps together)

new algorithm

while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set

improving the pruning step

new algorithm while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set

requirement

- cannot have subsumption chains, e.g. $C_1 \preceq C_2 \preceq C_3$

pre-processing replace chains by endpoint subsumptions (e.g. $C_1 \preceq C_3$)
computing adequate permutation

$\rightsquigarrow$ don't care how, they will be checked anyway!

improving the pruning step

new algorithm

while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set

optimizations

- provide C's in the order they were generated
  (replaces quadratic step by linear)

improving the pruning step

new algorithm

while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set

optimizations

- provide C's in the order they were generated
  (replaces quadratic step by linear)
- store Cs in a search tree
  (improves performance)

improving the pruning step

new algorithm

while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set

optimizations

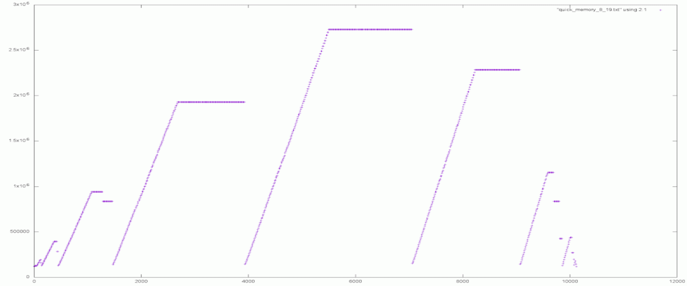
- provide C's in the order they were generated
  (replaces quadratic step by linear)
- store Cs in a search tree
  (improves performance)
- use search trees in some other places
  (duh?)

improving the pruning step

new algorithm while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set



improving the pruning step

new algorithm

while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set

less memory

- extract naturals to native integers
  (unfortunately necessary, but clearly sound)

improving the pruning step

new algorithm

while oracle has a next subsumption $C \preceq_\pi C'$

1. check that $C \preceq_\pi C'$
2. store C
3. remove C' from the current set

after: check that all stored networks are in the final set

less memory

- extract naturals to native integers
  (unfortunately necessary, but clearly sound)
- represent comparators as a single number
  (halves memory consumption)

philosophical considerations

the good news checker verifies s_9 in around 6 days using "moderate" resources

moderate not-so-new commonplace cpu, 64 gb ram

philosophical considerations

the good news checker verifies s_9 in around 6 days using "moderate" resources

more good news (almost) no changes to the formalization

- relatively quick changes (a few hours each)
- mostly require proving that optimized version coincides with original version

philosophical considerations

the good news checker verifies s_9 in around 6 days using "moderate" resources

more good news (almost) no changes to the formalization

- relatively quick changes (a few hours each)
- mostly require proving that optimized version coincides with original version

offline oracles a new methodology?

outline

conclusions & future work

results

- formal verification of exact values of s_n for $n \leq 9$
- new methodology (offline oracles)
- able to deal with $\sim$ 27 gb of proof witnesses
- clean separation between formalization ("mathematics") and optimization of checker ("computer science")

next episodes

- formal proof of van voorhis' $s_n \geq s_{n-1} + \lg(n)$ to obtain s_{10}
- other problems in sorting networks
- application of this method to other search-intensive proofs

thank you!